Postel

Edwin W. Meyer, Jr.
MIT Project MAC
April 25, 1970

(Both my personal opinions and those that I believe represent a consensus of the Network Working Group at Project MAC are presented here. The pronouns "I" and "we" are used to disinguish between these.)

On April 21 and 23 Thomas F. Skinner and I had telephone conversations with Steve Crocker at UCLA relating to the network protocol, specifically regarding our proposal in NWG/RFC 46. The following items were discussed. (I hope that Steve will pardon me if I happen to misparaphrase him.)

1) Steve stated that he felt that a need for dynamic reconnection would later be recognized by the network participants. However, because of a lack of consensus, it will not be included in the initial implementation. (We at Project MAC favor this approach of not including it initially.)

2) Steve supported the implementation of the INT network command described in NWG/RFC 46.

This command allows a process that has agreed to accept interrupts over a socket connection to be reliably interrupted by the process at the other end. The interrupt causes a process to obey its current execution and execute a procedure that it has specified as the INT handler. (The NCP does not specify the INT handler. That is the function of higher level protocols.)

The INT command is designed specifically for use by a third level User Control and Communication (UCC) protocol to implement a "quit" signal. Under such a protocol, both the requestor and the created process agree that an INT related to a specific socket connection and transmitted over the NCP control link to the created process is the standard "quit" signal. The created process provides an INT handler that implements this "quit" function. (This does not preclude a different interpretation of INT by other third level protocols.)

Although many systems implement the "quit" as a control character in the Teletype input stream, systems such as CTSS, Multics, and others implement it as a 200 ms spacing on the line. We at MAC think that the first method is an undesirable implementation within the network (while the second is impossible). I put forth several reasons why (and I think Steve agreed).

(a) The link over which the quit character is to be transmitted may be blocked.

(b) While the interrupt is most effectively implemented within the NCP, it is undesirable for the NCP to place any particular structure on the data being transmitted. (See discussion below.) This would be required if the NCP were to scan a data stream for a control character.

(c) Scanning the input stream greatly reduces NCP efficiency in a subsystem where speed is critical to effective operation.

Steve pointed out that the implementation of INT as a "quit" should not necessarily preclude a HOST's interpretation of a control character in the input stream from also acting as a "quit".

3) Steve is opposed both to including the instance tag in the socket identifier and reserving a null field in the identifier for future definition. He cited several reasons:

(a) Multiple processes of a single user should be indistinguishable to a foreign process. (I agree with this in certain cases when processes are co-ordinated in joint action. But what about the case where two processes of the same user both want to independently use the network?)

(b) A process wishing to connect to one of a foreign user's processes does not know the instance tag of the particular process that he wants, and he can't easily find out.

(c) If an instance tag should later prove desirable it could be added with some difficulty. (I claim that something as fundamental as the length of a socket identifier will prove very resistant to change.)

Tom stated that perhaps the low order three bits of the user code could be reserved for later interpretation as an instance tag. He doesn't think that a separate field is of great importance.
Steve's arguments seem to have merit. Perhaps Tom's suggestion is the way to go. I am currently undecided on this matter.

4) We all (Steve and MAC) seem to agree that at the NCP level there should be no special structure imposed on the data transmitted. To an NCP all data to be transmitted are bit strings of arbitrary length. One happy result is that the difficult question of character sets does not have to be resolved at this protocol level. To include a character set specification at the NCP level would delay agreement on the protocol and make this character set more resistant to change. (If there is to be

a standard character set. We prefer ASCII. After all, it is the prefered standard of our sponsoring organization.)

We also agree with Steve that there should be no optional echoing of messages at the NCP protocol level. (This is also the position of the SDC people in RFC 44.)

5) Shoshani, Long, and Landsberg also state (RFC 44) that they prefer to align messages to end on a word boundary as opposed to double padding. Steve agrees with us in not liking double padding.

6) In our proposal (RFC 46) we suggest that RFCs be queued only for open sockets, that RFCs to inactive or connected sockets are to be automatically rejected via the CLS command. Steve proposes that RFCs to these sockets be briefly queued. If the socket remains in an unacceptable state for a specific interval after the RFC comes in, it is rejected. This scheme allows certain types of network command interaction involving critical races to be implementable. Such a scheme of limited queueing does not seem unreasonable to me.

7) Steve, Tom, and I discussed strategies for a User Control and Communication (UCC) protocol. Steve said that he disliked our UCC strategy (RFC 46) because it requires maintaining two full-duplex connections to the requestor process and switching between them.

Steve put forth an alternate proposal: a process wishing to create a user process at a foreign HOST issues RFCs to sockets 0 and 1 belonging to the user whose process he wishes to create. If these sockets are inactive, the NCP automatically directs these requests to the foreign HOST's logger process. The logger accepts connection and performs the login ritual. If successful, the logger creates a user process and lets go of the usurped sockets so that the created process may use them to communicate with the requestor process. (I note that this does not use reconnection at a network level, since the logger uses sockets belonging to the ultimate user. However, it does involve internal reconnection.)

Tom and I objected to this because it introduces UCC protocol into the NCP level. (The NCP must direct all RFCs to inactive sockets 0 and 1 to a logger process.) I made a quick suggestion that perhaps our two proposals could be combined such that the requestor issues a "Signalling" RFC to a "signal" socket of the UCC process. The UCC rejects the RFC but remembers who is calling. It then tries to connect two sockets of the process to be created to the requestor's sockets, and conducts the login ritual through these. Steve liked this and suggested that I write it up.

Following the conversation, I thought of several disadvantages to this UCC strategy:

(a) If the control sockets at a created process are limited to 0 and 1, there is the possibility that a rightful user may not be able to communicate with a foreign UCC because the UCC already is using those sockets to communicate with an imposter. The logger will discover this and turn off the imposter, but this is an aggravating security breach. A malicious process could issue simultaneous multiple requests to tie up the sockets and prevent access to a rightful user. A better solution is to allow any socket pair of the potential user process to act as the control path. This permits the UCC to conduct simultaneous interrogations of competing requestors.

(b) A disadvantage of both Crocker's and the combined UCC is that the user to be logged in is specified by supplying a socket belonging to a particular user. The logger must now make the additional check that the user it is logging in actually belongs to the socket pair it is talking over. This seems the reverse of the prefered process: to identify a user and then determine the user code for his socket identifiers.

(c) The user may not know the socket user code of the user he wishes to log in at the foreign HOST. (After all, there is no basic reason why the requestor and created processes should have the same user code so long as the requestor satisfies the foreign logger.)

(d) In the combined strategy, there is no way for the requestor to specify which socket user code it wants. The only assumption that the UCC can make is that the requestor process wishes to log in a process having the same socket user code as itself. (This may not seem very important, but I envision a scheme in which a local process exists to allow consoles attached to the local HOST to login at a foreign HOST without being logged in locally.)

(e) The idea of allowing a process to masquerade within the network as another process (even with the best of intentions) by using its socket user code introduces a potentially dangerous security breach. I think that it should be a basic protocol law that NO PROCESS WHATSOEVER may request or accept connections or transmit or receive data over a socket having a user code not its own. This does not apply to an NCP process which has responsibility for such transmission, nor does it prevent a priviliged process from closing or rejecting connections between a foreign process and another local process.

I still think that the UCC proposal we advanced in RFC 46 is a good workable scheme. It does not require socket reconnection (either expressly throughout the network or implicitly within an NCP), nor do any of the objections raised above apply. The only particular disadvantage I see is that it requires the requestor process to maintain and switch between two full-duplex connections. I don't see this as a serious hindrance. I would like the comments of the network participants on this point in particular.

Fortunately the UCC is a third level protocol. The second level NCP can be specified before we reach final agreement on a UCC, provided that the NCP allows implementation of a workable UCC.

Steve expressed the thought that there need not be an initial standard UCC, that there might be several UCCs. We at MAC disagree. If we are all to talk to each other, and not between limited subsets of HOSTs within the network, there must be an initial standard UCC which EVERYBODY implements. (Steve is of course correct that there can be other experimental UCCs also implemented.)

It is theoretically possible for each HOST to provide multiple sets of software to allow a requestor process to communicate with the loggers at HOSTs implementing different UCCs. I don't think that it will work this way in practice. Each HOST will implement the UCC protocol that is most agreeable to it, and will provide one set of software so that a requestor process can communicate only with those HOSTs which implement similar UCCs.

I don't think that there is much enthusiasm at Project MAC for implementing a non-standard UCC just so we can talk to ourselves. We want to implement a single UCC supported at all installations, so that we can log in to all HOSTs using this protocol, and that users at all foreign HOSTs can log in to us.